Assignment 2

Huzaifa Jahan (215668106)
huzaifaj@my.yorku.ca

March 1, 2021

Note: You have to work individually. You must use the same mathematical notations in textbook or lecture slides to answer these questions. You must use this latex template to write up your solutions. Remember to fill in your information (name, student number, email) at above. No handwriting is accepted. In this assignment, you need to use the *MNIST* data set. Refer to

`https://colab.research.google.com/drive/1FyahMGAE22716sUCrNXPvTrKTwS615Hd`

for how to load it in Python. Direct your queries to Hui Jiang (`hj@eecs.yorku.ca`)

Exercise 1

Dimension Reduction

1. (5 marks) **PCA**: Q4.2 on page 93
2. (5 marks) **LDA**: Q4.4 on page 93
3. (10 marks) **Data visualization**: Lab Project I on page 92, parts a), b) and c)
   Note that you will have to implement PCA and LDA from scratch but you may choose to use a t-SNE implementation from any Python package.

Your answers:
1. PCA formula -

$$\frac{1}{N}\sum_{n=1}^{N}\|\mathbf{x}_n-\tilde{\mathbf{x}}_n\|$$

Next we will reprsent the points in a orthonormal basis this is the key of the entire proof:

$$\left\{\mathbf{u}_i:\mathbf{u}_i^T\mathbf{u}_i=1\right\}_{i=1}^{D}\quad \mathbf{u}_i^T\mathbf{u}_j=\begin{cases}1 & \text{if } i=j\\ 0 & \text{otherwise}\end{cases}$$

$$\begin{aligned}\mathbf{x}_n-\tilde{\mathbf{x}}_n&=\sum_{i=1}^{D}\left(\mathbf{x}_n^T\mathbf{u}_i\right)\mathbf{u}_i-\sum_{i=1}^{M}\left(\mathbf{x}_n^T\mathbf{u}_i\right)\mathbf{u}_i-\sum_{i=M+1}^{D}b_i\mathbf{u}_i\\ &=\sum_{i=M+1}^{D}\left(\mathbf{x}_n^T\mathbf{u}_i\right)\mathbf{u}_i-\sum_{i=M+1}^{D}b_i\mathbf{u}_i\\ &=\sum_{i=M+1}^{D}\left(\mathbf{x}_n^T\mathbf{u}_i\right)\mathbf{u}_i-b_i\mathbf{u}_i\end{aligned}$$

1.1

Now we have $b_i=\bar{x}^T\mathbf{u}_i$, thus the only thing left to minimize is $\mathbf{u}_i$.

$\frac{1}{N}\sum_{n=1}^{N}\left\|\sum_{i=M+1}^{D}\left(\left(\mathbf{x}_n-\tilde{\mathbf{x}}\right)^T\mathbf{u}_i\right)\mathbf{u}_i\right\|^2$

$=\frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\mathbf{u}_i^T\left(\mathbf{x}_n-\tilde{\mathbf{x}}\right)\left(\mathbf{x}_n-\tilde{\mathbf{x}}\right)^T\mathbf{u}_i$

$=\sum_{i=M+1}^{D}\mathbf{u}_i^T\mathbf{S}\mathbf{u}_i$

$=\sum_{j=m+1}^{D} \lambda_j$

2. LDA - From the point in the textbook where it finds the constraint:

$$\mathbf{w}^T S_W \mathbf{w} = 1$$

Now applying the Lagrange multipliers:

$$\phi(\mathbf{w}, \lambda) = \mathbf{w}^T S_B \mathbf{w} - \lambda \left(\mathbf{w}^T S_W \mathbf{w} - 1 \right)$$

Now differentiating with respect to w:

$$\frac{\partial \phi}{\partial \mathbf{w}} = 2 S_B \mathbf{w} - \lambda 2 S_W \mathbf{w} = 0$$
$$S_B \mathbf{w} = \lambda S_W \mathbf{w}$$

At this point in the problem we can now develop a generalized eigen-vector that is equivalent:

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w}$$

where λ and $\mathbf{w}$ are the eigenvalues and eigen-vectors of $S_W^{-1} S_B$ respectively. $\mathbf{w}$ is the largest eigen-vector (the eigen-vector corresponding to the largest eigenvalue) of $S_W^{-1} S_B$.

$$S_w^{-1} S_b w = \lambda w$$
$$S_w^{-1} (\mu_1 - \mu_0) (\mu_1 - \mu_0)^T w = \lambda w \quad \text{(based on } S_b\text{)}$$
$$S_w^{-1} (\mu_1 - \mu_0) = \frac{\lambda}{\alpha} w \quad \text{(where } \alpha = (\mu_1 - \mu_0)^T w \text{ is a scalar)}$$
$$S_w^{-1} (\mu_1 - \mu_0) \propto w'$$

2.1

Exercise 2

Linear Models for Regression

1. (10 marks) derive the formula to compute the gradients for the following linear models:
   (a) linear regression
   (b) ridge regression
   (c) LASSO

   follow the style of **Algorithm 2.3** (refer to `https://www.overleaf.com/learn/latex/algorithms`) to derive mini-batch stochastic gradient descent algorithms to optimize these models.

2. (20 marks) implement these three algorithms on a small data set, e.g. the Boston Housing Dataset (`https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html`), to predict median value of a home from the 13 attributes. You need to experimentally compare these regression models, discuss your results in terms of how the learning objective function and all learned model weights may differ among three models.

Your answers:
(A) Linear regression:(non-Multivariate Case) $Y = mX + b$ is key as that is the premise of Linear regression so this can be translated as our cost function. We require the cost function as

it is how the gradients can be further derived. Converting cost as mean squared error we get the following function where N is the total of observations, and y_i is the value of the of the observation seen. $MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (mx_i + b)\right)^2$ Derive function(very simple using chain rule the computation yields the 2 below) and compute the gradient with the inputs(since bias(b) and weight(m) are changeable only), with this mini batch SGD code below.

$$\frac{\partial}{\partial \mathrm{m}} = \frac{2}{N}\sum_{i=1}^{N} -x_i\left(y_i - (mx_i + b)\right) \quad \boxed{3.1}$$
$$\frac{\partial}{\partial \mathrm{b}} = \frac{2}{N}\sum_{i=1}^{N} -\left(y_i - (mx_i + b)\right)$$

(B) Ridge Regression: For ridge regression(1) unlike Linear we introduced the concept of a regularization parameter. (If the formula looks a bit different from linear it is because we used the simpler linear regression model without W_i). With this new parameter it allows our cost function to now penalize the size of weights. However, what makes this very special is that ridge regression can still be derived fairly easily this allows us to compute the gradients like in linear which results in (2). Similarly, we can optimize this with SGD with one change now being our hyper parameter. We must first find the best λ value by doing an MSE on the validation set. So the primary change to the algorithm would just be the implementation of a line that calculates the MSE using different λ values.

$$\text{(1).}\quad \mathbf{J}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(\left(W_0 + W_1X_1^{(i)} + \ldots + W_PX_P^{(i)}\right) - Y_i\right)^2 + \frac{\lambda}{2N}\sum_{j=1}^{P} W_j^2 \quad \boxed{3.2}$$
$$\text{(2).}\quad \frac{\partial \mathbf{J}(\mathbf{W})}{\partial W_k} = \frac{1}{N}\sum_{i=1}^{N}\left(\left(W_0 + W_1X_1^{(i)} + \ldots + W_PX_P^{(i)}\right) - Y_i\right)X_k^{(i)} + \frac{\lambda}{N}W_k$$

(C) LASSO: Lasso is a bit unique when contrasted to ridge due to the fact that the derivative of the cost function in lasso has NO CLOSED FORM. However the trick being that the Lasso penalty defines a boundary on our parameter space, and since the L_1 function is piece wise-linear, this allows the boundary to conveniently just be piece wise-linear. So we would initially start inside our boundary, do our gradient descent, and if/once we hit the boundary. We would know we are on a hyper-plane. This lets us line-search along the boundary, checking for the non deferential "corners" of the boundary (where a coordinate goes to zero). This is all possible because of the fact that Lasso allows for the chance too make a coefficient forced to zero.When doing Lasso compared to Ridge though we would be utilizing L_1 regularization instead of L_2 as in ridge. They key here in easier terms is just that instead of developing the first term we can develop the second as noted in (1). When doing the SGD Algorithm on Lasso we would need to of course just change the Loss function for it and just like ridge utilize the new parameters from the derivations to calculate.

$$\mathbf{J}(\mathbf{W}) = \frac{1}{2N}\sum_{i=1}^{N}\left(\left(W_0 + W_1X_1^{(i)} + \ldots + W_PX_P^{(i)}\right) - Y_i\right)^2 + \frac{\lambda}{2N}\sum_{j=1}^{P}\left|W_j\right|$$

$$\text{(1).}\quad \sum_{j=1}^{p}\frac{\partial}{\partial W_k}\left|W_j\right| \quad \boxed{3.3}$$
$$= \begin{cases} 0, & \text{if} \quad k = 0 \\ \frac{\partial |w_k|}{\partial w_k}, & \text{if} \quad k \neq 0 \end{cases}$$

MINI BATCH SGD ALGORITHM:

Initialize $\mathbf{w} := 0^{s-1}, b := 0$

for iteration $t \in [1, \ldots, T]$

for $i \in [1, \ldots, s]$ (where s represents the mini-batch size)

Draw random example with replacement: $\left\langle \mathbf{x}^{[i]}, y^{[i]} \right\rangle \in \mathcal{D}$

Loss $\mathcal{L} := \frac{1}{s}\sum_{i=1}^{s} L\left(\hat{y}^{[i]}, y^{[i]}\right)$

Gradient $\Delta \mathbf{w} := -\nabla_{\mathcal{L}} \mathbf{w}, \Delta b := -\frac{\partial \mathcal{L}}{\partial}$
Update parameters $\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}, b := +\Delta b$

2. Since my codes would not work perfectly I did implement some with the use of libraries to at least understand the component of how the regression models compare, and how the learning objective function and all learned model weights may differ among three models. The linear regression model was weaker compared to the other two. One issue with linear regression was that it does not account for the chance of over fitting is what I noted. Ridge regression however does, it takes care of this by shrinking certain parameters as seen in the math even before further proofing it. Lasso on the other hand takes the concept of ridge even further by allowing us the ability to make certain coefficients forced to zero, eliminating them from the models. Specifically when comparing linear to ridge we noted two things the MSE decreases and ridge regression ended up squeezing the coefficients for X's close to zero. Now when comparing Lasso to the same result was seen how that LASSO pushes the coefficients to exact zero for multiple features now from the Boston-data set. Since our data set has the problem of Multicollinearity the lasso and ridge models performed better in this regard when compared to linear for more precision. This led me to deduct the true importance of the regularization term.

Exercise 3

Support Vector Machine (SVM)

1. (10 marks) Q6.8 on page 130
2. (20 marks) use all training data of two digits '5' and '8' from the MNIST dataset to learn two binary classifiers using linear SVM and nonlinear SVM (with Gaussian RBF kernel), and compare and discuss the performance and efficiency of linear SVM and nonlinear SVM methods for these two digits. Report your best results in the test data of '5' and '8'. Don't call any off-the-shelf optimizer. Use the projected gradient descent in Algorithm 6.5 to implement the SVM optimizer yourself.

Your answers:

1.

Since we know that: $\mathbf{x^T x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix} = \begin{bmatrix} x_1^2 & x_1 x_2 & \dots & x_1 x_m \\ x_1 x_2 & x_2^2 & \dots & x_2 x_m \\ \dots & \dots & \ddots & \dots \\ x_1 x_m & x_2 x_m & \dots & x_m^2 \end{bmatrix}$ now we have: $\mathbf{x x^T} = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \sum_{i=1}^{m} \mathbf{x}_i^2 = \sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_j^T$

4.1

The same principal applies to $\sum_{i=1}^{m} \mathbf{x}_i \mathbf{y}_j^T$, thus proving that both the following matrix multiplications can be vectorized This concept is necessary as the Q matrix calculation for SVM machines that are non-linear requiring kernel functions utilizes these transpose properties as it helps simplify the process combined with the kernel trick. So we could simplify the Q formula from the textbook. Utilizing the transpose property it allows us to rewrite the $\left[\mathbf{y y}^\top\right]_{N \times N}$ matrix with the above property $\mathbf{x x^T}$ and also utilizing the same trick for the kernel matrix we can simplify it as the first property $\mathbf{x^T x}$. Also understanding the nature of the kernel function allows us to remove it as well further iterating down below.

An example: Utilizing this polynomial non-linear basis function. We can see that with the kernel matrix every term will be an inner product of 2 samples. When expanding the terms, we see that the inner product terms of non linear basis functions, can be seen as a function of the BASIS VECTOR inputs instead, when grouped up. This results in an efficient way by using the kernel functions to compute the Q matrix as we have now seen that. We can find the kernel matrix without needing to know the nature of ϕ (All done thru the use of the kernel trick)(SHOULD BE mentioned that when applying this specifically to the Gaussian kernel saw how the middle factor further expands as a power series since polynomial and linear are showed as with the example).

$$\phi : x \to \phi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \qquad \begin{aligned} \phi\left(x_i\right)^T \phi\left(x_j\right) &= \left(x_{i1}x_{j1} + x_{i2}x_{j2}\right)^2 \\ \phi\left(x_i\right)^T \phi\left(x_j\right) &= \left(x_i^T x_j\right)^2 \\ \phi\left(x_i\right)^T \phi\left(x_j\right) &= k\left(x_i, x_j\right) \end{aligned}$$

5.1

$$\phi\left(x_i\right)^T \phi\left(x_j\right) = \begin{bmatrix} x_{i1}^2 \\ \sqrt{2}x_{i1}x_{i2} \\ x_{i2}^2 \end{bmatrix}^T \begin{bmatrix} \sqrt{2}x_{j1}x_{j2} \\ x_{j2}^2 \end{bmatrix}$$

$$\phi\left(x_i\right)^T \phi\left(x_j\right) = x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2$$

$$\text{Polynomial: } k\left(x_i, x_j\right) = \left(x_i^T x_j + r\right)^d$$

$$\text{Gaussian: } k\left(x_i, x_j\right) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

$$\text{Linear: } k\left(x_i, x_j\right) = \left(x_i^T x_j\right)$$

5.2

2. Once again could not properly implement without libraries so I still utilized the library to gauge the component of comparing and discussing the performance and efficiency of linear SVM and nonlinear with the MNIST - data set. The results of the linear vs non-linear was pretty obvious as linear percentage for correctness was lower than the Gaussian kernel even without tuning the hyper parameters after tuning the parameters it was more efficient and faster than linear. This led me to the conclusion that the problem is non-linear in nature which also makes sense from a theoretical stand point.

What to submit?

You must submit:

1. one PDF document (using this latex template) for your solutions to all written questions and all results and discussions for your programming assignments
2. one zip file that includes all of your Python codes and a readme file for TA to run your codes

from eClass before the deadline. No late submission will be accepted.

Index of comments